Data process (code from A1)

```
import pandas
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim

from nltk.stem import PorterStemmer
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
```

Download data to txt file

```
import os

# plz edit path
file_path = "/Users/wangzeyang/Desktop/CSC413/A2/aclImdb/test/neg"
file_path_pos = "/Users/wangzeyang/Desktop/CSC413/A2/aclImdb/test/pos"

lst = os.listdir(file_path)
lst_2 = os.listdir(file_path_pos)
print(lst_2)
new_file = open("valid_data.txt", "w+")

for file_name in lst:
    read_file = open(file_path + "/" + file_name, "r+")
    new_file.write(file_name[-5] + " ")
    content = read_file.read(1600)
    new_file.write(content)
    new_file.write("\n")
    read_file.close()

for file_name in lst_2:
    read_file = open(file_path_pos + "/" + file_name, "r+")
    new_file.write(file_name[-5] + " ")
    content = read_file.read(1600)
    new_file.write(content)
    new_file.write("\n")
    read_file.close()

new_file.close()
```

✓ 12m 46s completed at 7:36 PM

```
file_path = "/Users/wangzeyang/Desktop/CSC413/A2/aclImdb/train/neg"
file_path_pos = "/Users/wangzeyang/Desktop/CSC413/A2/aclImdb/train/pos"

lst = os.listdir(file_path)
lst_2 = os.listdir(file_path_pos)
print(lst_2)
new_file = open("train_data.txt", "w+")

for file_name in lst:
    read_file = open(file_path + "/" + file_name, "r+")
    new_file.write(file_name[-5] + " ")
    content = read_file.read(1600)
    new_file.write(content)
    new_file.write("\n")
    read_file.close()

for file_name in lst_2:
    read_file = open(file_path_pos + "/" + file_name, "r+")
    new_file.write(file_name[-5] + " ")
    content = read_file.read(1600)
    new_file.write(content)
    new_file.write("\n")
    read_file.close()

new_file.close()
```

Read data to colab

```
# upload dataset
from google.colab import drive
drive.mount('/content/gdrive')


train_path = '/content/gdrive/My Drive/CSC413/train_data.txt'
valid_path = '/content/gdrive/My Drive/CSC413/valid_data.txt'
```

Part 1 Data anlyze

1-a: Functions for data analyze.

```
import string
import collections

def get_score(path):
```

```
    data = []

    for line in open(path):
        score = int(line.replace("<br />", "").split()[0])
        data.append(score or 10)

    return data

def read_sentences1(path, max_word):
    data = []
    punct = string.punctuation

    for line in open(path):
        # replace <br/> and punctuations
        line_1 = line.replace("<br />", "")
        # remove punctuations
        for i in line_1:
            if i in punct:
                line_1 = line_1.replace(i, " ")
        words_1 = line_1.split()
        words = [word for word in words_1 if len(word) > 3]

        # set maximum length for each sentence and pad with <EOS>
        if len(words) > max_word:
            sentence = [word.lower() for word in words[0 : max_word]]
        else:
            sentence = [word.lower() for word in words[0:]]
        data.append(sentence)

    return data

def read_sentences2(path, max_word):
    data = []
    punct = string.punctuation
    stopWords = stopwords.words("english")
    stemmer = PorterStemmer()

    for line in open(path):
        # replace <br/> and punctuations
        line_1 = line.replace("<br />", "")
        # remove punctuations
        for i in line_1:
            if i in punct:
                line_1 = line_1.replace(i, " ")
        words_1 = line_1.split()
        # remove stop words
        words = [word for word in words_1 if len(word) > 3 and word not in stopWorc
        for i in range(len(words)):
            words[i] = stemmer.stem(words[i])
        for word in words:
            for l in range(len(word) - 2):
```

```
                if word[l] == word[l + 1] == word[l + 2]:
                    words.remove(word)
                    break
        for word in words:
            for l in word:
                if l.isdigit():
                    words.remove(word)
                    break

        if len(words) > max_word:
            sentence = [word.lower() for word in words[0 : max_word]]
        else:
            sentence = [word.lower() for word in words[0:]]
        data.append(sentence)

    return data

def word_anaylze(data):
    all_words = [word for sentence in data for word in sentence]

    word_counts = collections.Counter(all_words)

    print('Most common 10 words:')
    print(word_counts.most_common(10))
    vocabulary_size = len(word_counts)
    print("size of vocab: ", vocabulary_size)

    sentence_lengths = [len(sentence) for sentence in data]
    sentence_length_counts = collections.Counter(sentence_lengths)
    print("sentence length: ", sentence_length_counts)
    return
```

1-b. Analyze the score part

```
import statistics
score_data = get_score(train_path)



print("total number of scores:", len(score_data))

occur_counts = []
for i in set(score_data):
    occur_counts.append(count:= score_data.count(i))
    print(i, "occurrence count: ", count)

print("negative sum: ", sum(occur_counts[:4]))
print("positive sum: ", sum(occur_counts[4:]))
```

```
print("mean: ", statistics.mean(score_data))
print("median: ", statistics.median(score_data))
print("most occurrence number: ", statistics.mode(score_data))
```

1-c. Analyze the word part

```
max_word = 100
words_data = read_sentences1(train_path, max_word)
words_data2 = read_sentences2(train_path, max_word)
```

```
print('before remove stop words: ')
word_anaylze(words_data)
print('after remove stop words: ')
word_anaylze(words_data2)
```

Part 2 Data processing

```
import string

def read_sentences(path, max_word):
    data = []
    punct = string.punctuation
    stopWords = stopwords.words("english")
    stemmer = PorterStemmer()

    for line in open(path):
        # replace <br/> and punctuations
        line_1 = line.replace("<br />", "")
        # remove punctuations
        for i in line_1:
            if i in punct:
                line_1 = line_1.replace(i, " ")
        words_1 = line_1.split()
        # remove stop words
        words = [word for word in words_1 if len(word) > 4 and word not in stopWord
        # reduce to root
        for i in range(len(words)):
            words[i] = stemmer.stem(words[i])
        # remove incorrect spelling words (words with 3 same letters)
        for word in words:
            for l in range(len(word) - 2):
```

```
                if word[l] == word[l + 1] == word[l + 2]:
                    words.remove(word)
                    break
        # remove digits
        for word in words:
            for l in word:
                if l.isdigit():
                    words.remove(word)
                    break

        # set maximum length for each sentence and pad with empty string
        if len(words) > max_word:
            sentence = [words_1[0], [word.lower() for word in words[0 : max_word]]]
        else:
            sent = [word.lower() for word in words[0:]]
            for i in range(len(words), max_word):
                sent.append("")
            sentence = [words_1[0], sent]
        data.append(sentence)

    return data


# read sentences and vocab
max_word = 100
train_data_1 = np.array(read_sentences(train_path, max_word))
valid_data_1 = np.array(read_sentences(valid_path, max_word))


print(len(train_data_1))

vocab = set([w for s in train_data_1 for w in s[1]])

vocab_size = len(vocab)

print(vocab_size) # 41456


# split validation and test set 50%
splt_1 = 0
splt_2 = 12500
np.random.shuffle(valid_data_1)
np.random.shuffle(train_data_1)


train_data = np.concatenate((train_data_1, valid_data_1[0:splt_1]), axis=0)
valid_data = valid_data_1[splt_1:splt_1 + splt_2]
test_data = valid_data_1[splt_1 + splt_2:]

print(len(train_data)) # 30000
print(len(valid_data)) # 10000
```

```
print(len(test_data))  # 10000


# test if same number of positive and negative reviews in validation set
total_1 = 0
total_2 = 0
for i in valid_data:
    if int(i[0]) > 5 or int(i[0]) == 0:
        total_1 += 1
    else:
        total_2 += 1

print (total_1)
print (total_2)
```

Input process

```
vocab_lst = sorted(list(vocab))
# move padding char '' (empty string) to index 0
vocab_lst.remove("")
vocab_lst.insert(0, "")
print(len(vocab_lst), vocab_lst)
# A mapping of index => word (string)
vocab_itos = dict(enumerate(vocab_lst))
# A mapping of word => its index
vocab_stoi = {word:index for index, word in vocab_itos.items()}

def convert_words_to_indices(sents):
    indices = []
    for i in range(len(sents)):
      ind = []
      for word in sents[i]:
        ind.append(vocab_stoi[word] if word in vocab_lst else vocab_stoi[""])
      indices.append(ind)
    return indices

def get_batch(data, range_min, range_max):
    """
    Convert one batch of data in the form of sentence into input and output
    data and return the training data (xs, ts) where:
     - `xs` is an numpy array of indices [batch_size, max_word]
     - `ts` is a numpy array of shape [batch_size] containing indicies

    Preconditions:
     - `data` is a numpy array of shape [N, 2]
     - range_max > range_min
    """
    xs, ts = [], []
```

```
    for d in data[range_min:range_max]:
        ts.append(int(d[0]))
        xs.append(d[1])
    xs = convert_words_to_indices(xs)
    xs = np.array(xs)
    ts = np.array(ts)

    return xs, ts
```

Part 3 Build model and train

```
## model

class LSTMmodel(nn.Module):
    def __init__(self, vocab_size = vocab_size, emb_size = 100, num_hidden = 150):
        super(LSTMmodel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.conv1 = nn.Conv1d(in_channels=emb_size, out_channels=32, kernel_size=3
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.lstm = nn.LSTM(50, num_hidden, batch_first=True)
        self.dropout = nn.Dropout(p=0.2)
        self.fc1 = nn.Linear(num_hidden, 50)
        self.fc2 = nn.Linear(50, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = x.long() # for embedding type debug
        x = self.embedding(x)
        x = self.dropout(x)
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.lstm(x)[0]
        x = x[:, -1, :]
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.softmax(x)
        return x
```

Train function

```
def train(model, train_data = train_data, valid_data = valid_data, batch_size=32, v
          learning_rate=0.001, max_iters=7, checkpoint_path=None):

  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(), lr=learning_rate,
```

```
                      weight_decay=weight_decay)

iters, losses = [], []
iters_sub, train_accs, val_accs  = [], [], []

n = 0 # the number of iterations
while True:
    for i in range(0, train_data.shape[0], batch_size):
        if (i + batch_size) > train_data.shape[0]:
            break

        # get the input and targets of a minibatch
        xs, ts = get_batch(train_data, i, i + batch_size)

        # convert from numpy arrays to PyTorch tensors
        xs = torch.Tensor(xs)
        ts = torch.Tensor(ts).long()

        zs = model(xs)
        loss = criterion(zs, ts) # compute the total loss
        loss.backward()          # compute updates for each parameter
        optimizer.step()         # make the updates for each parameter
        optimizer.zero_grad()    # a clean up step for PyTorch

        # save the current training information
        iters.append(n)
        losses.append(float(loss)/batch_size)  # compute *average* loss

        if n % 200 == 0: # was 500
            iters_sub.append(n)
            train_cost = float(loss.detach().numpy())
            train_acc = get_accuracy(model, train_data)
            train_accs.append(train_acc)
            val_acc = get_accuracy(model, valid_data)
            val_accs.append(val_acc)
            print("Iter %d. [Val Acc %.0f%%] [Train Acc %.0f%%, Loss %f]" % (
                  n, val_acc * 100, train_acc * 100, train_cost))

            if (checkpoint_path is not None) and n > 0:
                torch.save(model.state_dict(), checkpoint_path.format(n))

        # increment the iteration number
        n += 1

        if n > max_iters:
            return iters, losses, iters_sub, train_accs, val_accs
```

Accuracy

```
def get_accuracy(model, data, batch_size=32, max_N=100):
    """
    Estimate the accuracy of the model on the data. To reduce
    computation time, use at most `max_N` elements of `data` to
    produce the estimate.
    """
    correct = 0
    correct_1 = 0
    N = 0
    for i in range(0, data.shape[0], batch_size):
        # get a batch of data
        xs, ts = get_batch(data, i, i + batch_size)

        # forward pass prediction
        z = model(torch.Tensor(xs))
        z = z.detach().numpy() # convert the PyTorch tensor => numpy array
        pred = np.argmax(z, axis=1)
        correct += np.sum(pred == ts)

        for i in range(1, 5):
            for j in range(1, 5):
                pred_cur = pred[pred == i]
                ts_cur = ts[pred == i]
                pred_cur = pred_cur[ts_cur == j]
                correct_1 += pred_cur.shape[0]

        for j in range(6, 10):
            for i in range(6, 10):
                pred_cur = pred[pred == i]
                ts_cur = ts[pred == i]
                pred_cur = pred_cur[ts_cur == j]
                correct_1 += pred_cur.shape[0]

                pred_cur = pred[pred == i]
                ts_cur = ts[pred == i]
                pred_cur = pred_cur[ts_cur == 0]
                correct_1 += pred_cur.shape[0]

            pred_cur = pred[pred == 0]
            ts_cur = ts[pred == 0]
            pred_cur = pred_cur[ts_cur == j]
            correct_1 += pred_cur.shape[0]

        N += ts.shape[0]

        if N > max_N:
            break
    acc_1 = correct / N
    acc_2 = correct_1 / N
```

```
    return acc_2



def get_accuracy_1(model, data, batch_size=32, max_N=100):
    """
    Estimate the accuracy of the model on the data. To reduce
    computation time, use at most `max_N` elements of `data` to
    produce the estimate.
    """
    correct = 0
    correct_1 = 0
    N = 0
    pred_list = []
    for i in range(0, data.shape[0], batch_size):
        # get a batch of data
        xs, ts = get_batch(data, i, i + batch_size)

        # forward pass prediction
        z = model(torch.Tensor(xs))

        z = z.detach().numpy() # convert the PyTorch tensor => numpy array

        pred = np.argmax(z, axis=1)
        pred_list.append(pred)

    return np.array(pred_list)



def plot_learning_curve(iters, losses, iters_sub, train_accs, val_accs):
    """
    Plot the learning curve.
    """
    plt.title("Learning Curve: Loss per Iteration")
    plt.plot(iters, losses, label="Train")
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.show()

    plt.title("Learning Curve: Accuracy per Iteration")
    plt.plot(iters_sub, train_accs, label="Train")
    plt.plot(iters_sub, val_accs, label="Validation")
    plt.xlabel("Iterations")
    plt.ylabel("Accuracy")
    plt.legend(loc='best')
    plt.show()
```

Naive-bayes model

```
def read_sentences_naive(path, max_word):
    data = []
    punct = string.punctuation
    stopWords = stopwords.words("english")
    stemmer = PorterStemmer()

    for line in open(path):
        # replace <br/> and punctuations
        line_1 = line.replace("<br />", "")
        # remove punctuations
        for i in line_1:
            if i in punct:
                line_1 = line_1.replace(i, " ")
        words_1 = line_1.split()
        # remove stop words
        words = [word for word in words_1 if len(word) > 4 and word not in stopWord
        for i in range(len(words)):
            words[i] = stemmer.stem(words[i])
        for word in words:
            for l in range(len(word) - 2):
                if word[l] == word[l + 1] == word[l + 2]:
                    words.remove(word)
                    break
        for word in words:
            for l in word:
                if l.isdigit():
                    words.remove(word)
                    break

        sentence = [int(words_1[0]), [word.lower() for word in words[0 : max_word]]
        data.append(sentence)

    return data


# read sentences and vocab
max_word = 150
train_data_naive = np.array(read_sentences_naive(train_path, max_word))
valid_naive = np.array(read_sentences_naive(valid_path, max_word))

vocab = set([w for s in train_data_naive for w in s[1]])
vocab_size = len(vocab)
print(vocab_size)


# split valid data set into valid set and test set
splt = len(valid_naive) // 2
test_data_naive = valid_naive[splt :]
valid_data_naive = valid_naive[: splt]

print(len(train_data_naive)) # 25000
```

```
print(len(train_data_naive)) # 25000
print(len(valid_data_naive)) # 12500


print(train_data[0])


# Build the input of Naive-bayes model
train_sen = train_data_naive[:,1].tolist()
# print(train_sen[100])
t = train_data_naive[:,0]
t_list = t.tolist()
print(t_list)
t1 = np.zeros([len(t_list), 10])
for i in range(len(t_list)):
    t1[i][t_list[i]] = 1
print(t1)


print(len(train_sen))


# Build naive-bayes model
def make_matrix(data, vocab):
    num = 0
    true_vocab = {}
    for i in range(len(vocab)):
        true_vocab[vocab[i]] = i

    X = np.zeros([len(data), len(vocab)])
    for i in range(len(data)):
        list_vocab = data[i]

        for j in list_vocab:
            if j in vocab:
                X[i][true_vocab[j]] = 1
        num += 1
        print(num)

    return X

def naive_bayes_map(X, t, t1):
    N, vocab_size = X.shape[0], X.shape[1]
    t1_sum = t1.sum(0)
    pi = (t1_sum + 2 - 1) / (N + 2 + 2 - 2)
    theta = np.zeros([vocab_size, 10])

    X_1 = X[t == 1]
    X_2 = X[t == 2]
    X_3 = X[t == 3]
    X_4 = X[t == 4]
    X_5 = X[t == 5]
    X_6 = X[t == 6]
```

```
        X_6 = X[t == 6]
        X_7 = X[t == 7]
        X_8 = X[t == 8]
        X_9 = X[t == 9]
        X_10 = X[t == 0]

        N_1 = X_1.shape[0]
        X_1 = (X_1.sum(axis = 0) + 1) / (N_1 + 2)
        theta[:,0] = theta[:,0] + X_1

        N_2 = X_2.shape[0]
        X_2 = (X_2.sum(axis = 0) + 1) / (N_2 + 2)
        theta[:,1] = theta[:,1] + X_2

        N_3 = X_3.shape[0]
        X_3 = (X_3.sum(axis = 0) + 1) / (N_3 + 2)
        theta[:,2] = theta[:,2] + X_3

        N_4 = X_4.shape[0]
        X_4 = (X_4.sum(axis = 0) + 1) / (N_4 + 2)
        theta[:,3] = theta[:,3] + X_4

        N_5 = X_5.shape[0]
        X_5 = (X_5.sum(axis = 0) + 1) / (N_5 + 2)
        theta[:,4] = theta[:,4] + X_5

        N_6 = X_6.shape[0]
        X_6 = (X_6.sum(axis = 0) + 1) / (N_6 + 2)
        theta[:,5] = theta[:,5] + X_6

        N_7 = X_7.shape[0]
        X_7 = (X_7.sum(axis = 0) + 1) / (N_7 + 2)
        theta[:,6] = theta[:,6] + X_7

        N_8 = X_8.shape[0]
        X_8 = (X_8.sum(axis = 0) + 1) / (N_8 + 2)
        theta[:,7] = theta[:,7] + X_8

        N_9 = X_9.shape[0]
        X_9 = (X_9.sum(axis = 0) + 1) / (N_9 + 2)
        theta[:,8] = theta[:,8] + X_9

        N_10 = X_10.shape[0]
        X_10 = (X_10.sum(axis = 0) + 1) / (N_10 + 2)
        theta[:,9] = theta[:,9] + X_10

        return pi, theta

    def training_3(data, t, t1, vocab):
        X = make_matrix(data, vocab)
```

```
    pi, theta = naive_bayes_map(X, t, t1)
    return [pi, theta, vocab]


# Create prediction function
def make_prediction_3(review, vocab, pi, theta):
    punctuation = string.punctuation
    for punc in punctuation:
        review = review.replace(punc, "")
    X = np.zeros([1, len(vocab)])
    words = review.split()
    for j, w in enumerate(vocab):
        if w in words:
            X[0, j] = 1

    V, K = theta.shape

    X_opposite = np.ones([1, V]) - X
    theta_opposite = np.ones([V, K]) - theta

    cur_largest_pro = 0
    cur_large_k = 0
    for k in range(K):
        exist = X * theta[:,k]
        not_exist = X_opposite * theta_opposite[:,k]
        All = exist + not_exist
        cur_p = np.log(All)
        cur_p = cur_p.sum(1)
        cur_p = np.exp(cur_p)
        cur_p = cur_p * pi[k]
        if cur_p > cur_largest_pro:
            cur_largest_pro = cur_p
            cur_large_k = k

    return cur_large_k + 1


# get accuracy
def get_accuracy_naive(data_set, vocab, pi, theta):
    accurate = 0
    acc = 0
    for i in data_set:
        sentence = i[1]
        cur_sen = ""
        for word in sentence:
            cur_sen += word + " "
        pred = make_prediction_3(cur_sen, vocab, pi, theta)
        if pred == i[0]:
            accurate += 1
        if 0 < pred < 6 and 0 < i[0] < 6:
            acc += 1
```

```
            if 5 < pred < 11 and 5 < i[0] < 11:
                acc += 1

        return acc / len(data_set)


    def total_training_model(x, t, t1):
        model_total = []

        #training mode 1 -- RNN model

        model_1 = []

        lstm_model = LSTMmodel()
        learning_curve_info = train(lstm_model, max_iters=400)
        model_1.append(lstm_model)

        #training mode 2 -- Naive Bayes model

        model_3 = training_3(train_sen, t, t1, list(vocab))

        model_total.append(model_1)
        model_total.append(model_3)

        return model_total


    def total_predict_1(x, test, test_data, vocab, model_total):
        cur_sen = ""
        for word in test:
            cur_sen += word + " "
        y2 = make_prediction_3(cur_sen, list(vocab), model_total[1][0], model_total[1]|

        return y2


    def total_predict_2(x, test_data, vocab, model_total):

        y1 = get_accuracy_1(model_total[0][0], test_data)

        return y1


    model_total = total_training_model(train_data, t, t1)
```

Part 4 Results

```
result_list = []
target_list = []
num_here = -1
for i in test_data:
    num_here += 1
    result = total_predict_1(train_data, i[1], valid_data, vocab, model_total)
    target = i[0]
    result_list.append(result)
    target_list.append(target)
    print ("number:", num_here, "result:", result, "target:", target)

result_1 = np.array(result_list)
target_1 = np.array(target_list)
print (test_data[1])


result_2 = total_predict_2(train_data, valid_data, vocab, model_total)

print(result_2)


result_3 = np.array([])

for i in result_2:
    result_3 = np.concatenate((result_3, i), axis = 0)

print(result_3.shape)
print(target_1.shape)
```

```
(12500,)
(12500,)
```

```
result_input = np.vstack((result_1, result_3))
result_input = result_input.transpose()
print(result_input.shape)
```

```
(12500, 2)
```

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression(fit_intercept=False).fit(result_input, target_1)
print(lr.coef_)
```

```
[0.76630422 1.5897359 ]
```

```
def final_total_predict_1(x, test, test_data, vocab, model_total):
    cur_sen = ""
    for word in test:
```

```
        cur_sen += word + " "
    y2 = make_prediction_3(cur_sen, list(vocab), model_total[1][0], model_total[1]|

    return y2


def final_total_predict_2(x, test_data, vocab, model_total):

    y1 = get_accuracy_1(model_total[0][0], test_data)

    return y1


final_2 = final_total_predict_2(train_data, test_data, vocab, model_total)

final_3 = np.array([])

for i in final_2:
    final_3 = np.concatenate((final_3, i), axis = 0)

print(final_3.shape)
print(final_3)
```

```
(12500,)
[0. 0. 1. ... 0. 1. 0.]
<ipython-input-78-8b51ec17a987>:23: VisibleDeprecationWarning: Creating an nda
  return np.array(pred_list)
```

```
correct = 0
total = len(test_data)

for i in range(len(test_data)):
    result = total_predict_1(train_data, test_data[i][1], test_data, vocab, model_t
    target = test_data[i][0]
    if int(target) == 0:
        target = 10
    else:
        target = int(target)

    result = int(result)

    result = (lr.coef_[1] * result + lr.coef_[0] * final_3[i]) / 2
    result = int(result)

    if int(target) < 5 and int(result) < 5:
        correct += 1
```

```
            correct += 1

        if int(target) > 5 and int(result) > 5:
            correct += 1

        if i % 100 == 0 and i != 0:
            print(correct/i)

        print ("number:", i, "result:", result, "target:", target)

    print(correct/total)
```

```
Streaming output truncated to the last 5000 lines.
number: 7550 result: 1 target: 1
number: 7551 result: 8 target: 9
number: 7552 result: 6 target: 8
number: 7553 result: 8 target: 10
number: 7554 result: 1 target: 1
number: 7555 result: 0 target: 1
number: 7556 result: 1 target: 1
number: 7557 result: 8 target: 9
number: 7558 result: 3 target: 3
number: 7559 result: 1 target: 9
number: 7560 result: 0 target: 9
number: 7561 result: 1 target: 3
number: 7562 result: 8 target: 10
number: 7563 result: 8 target: 8
number: 7564 result: 7 target: 10
number: 7565 result: 7 target: 10
number: 7566 result: 1 target: 4
number: 7567 result: 8 target: 10
number: 7568 result: 1 target: 4
number: 7569 result: 1 target: 2
number: 7570 result: 1 target: 1
number: 7571 result: 2 target: 3
number: 7572 result: 1 target: 1
number: 7573 result: 1 target: 4
number: 7574 result: 1 target: 1
number: 7575 result: 0 target: 10
number: 7576 result: 1 target: 1
number: 7577 result: 3 target: 4
number: 7578 result: 1 target: 4
number: 7579 result: 6 target: 7
number: 7580 result: 3 target: 4
number: 7581 result: 1 target: 4
number: 7582 result: 7 target: 7
number: 7583 result: 1 target: 2
number: 7584 result: 0 target: 4
number: 7585 result: 6 target: 8
number: 7586 result: 8 target: 8
number: 7587 result: 2 target: 4
number: 7588 result: 0 target: 10
number: 7589 result: 8 target: 9
number: 7590 result: 6 target: 3
```

```
number: 7591 result: 7 target: 3
number: 7592 result: 6 target: 10
number: 7593 result: 1 target: 2
number: 7594 result: 7 target: 1
number: 7595 result: 1 target: 7
number: 7596 result: 1 target: 7
number: 7597 result: 7 target: 7
number: 7598 result: 0 target: 8
number: 7599 result: 8 target: 4
0.8243421052631579
number: 7600 result: 6 target: 8
number: 7601 result: 8 target: 8
number: 7602 result: 8 target: 8
number: 7603 result: 8 target: 7
number: 7604 result: 0 target: 4
number: 7605 result: 1 target: 1
```